United States Patent [19]

Carron et al.

[11] Patent Number: 4,724,521

[45] Date of Patent: Feb. 9, 1988

[54] METHOD FOR OPERATING A LOCAL TERMINAL TO EXECUTE A DOWNLOADED APPLICATION PROGRAM

[75] Inventors: James M. Carron, Aiea; Brian K. Uechi; Mohammed A. Khan, both of Honolulu; Clifton W. Royston, III; Jay A. Abel, both of Honolulu; Bradley J. Ferlane; Robert K. L. Loui, both of Honolulu; William R. Pape, III, Papaaloa, all of Hi.

[73] Assignee: Veri-Fone, Inc., Honolulu, Hi.

[21] Appl. No.: 819,186

[22] Filed: Jan. 14, 1986

[51] Int. Cl.⁴ G06F 15/16

[52] U.S. Cl. 364/300; 364/200

[58] Field of Search .... 364/200 MS File, 300

[56] References Cited

U.S. PATENT DOCUMENTS

4,459,662 7/1984 Skelton et al. 364/200

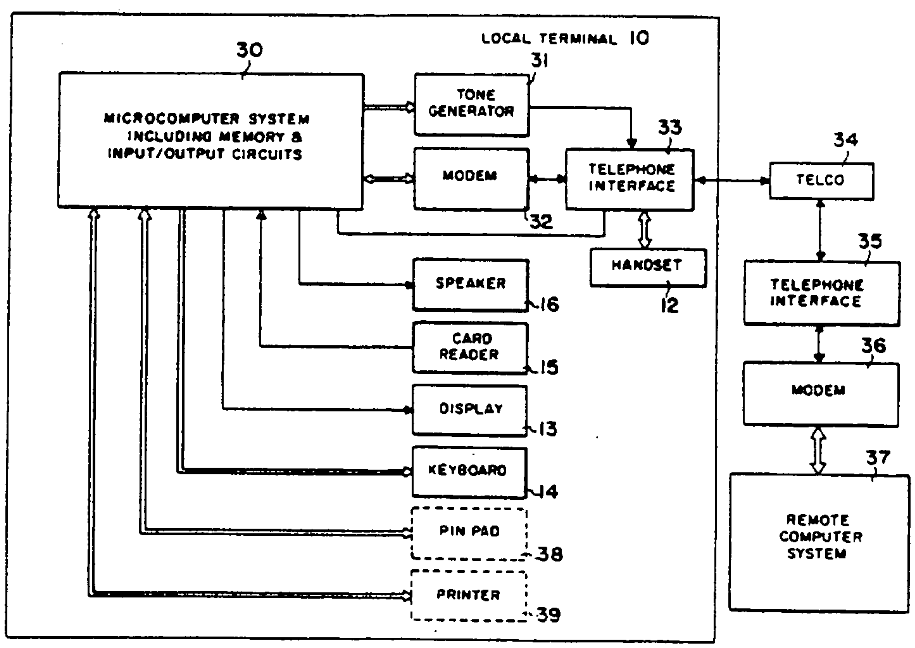
Primary Examiner—Raulfe B. Zache

Attorney, Agent, or Firm—Lowell C. Bergstedt

[57] ABSTRACT

The present invention provides methods for operating a local terminal which includes a programmable computer so that the terminal executes a pre-arranged application program. More specifically, the present invention provides methods for operating a local terminal according to a pre-arranged application program which is created on a remote computer, then communicated by a communication channel to the local terminal where it is stored for execution.

18 Claims, 65 Drawing Figures



What is claimed is:

1. In a method for operating a local terminal which includes a local computer system comprising a central processor unit, a read only memory coupled to said central processor unit, a random access memory coupled to said central processor unit, and a communication channel coupled to said central processor unit for communicating with a remote computer system, the steps of:

a. establishing a set of general purpose operation routines to be executed by said local computer system, each of said general purpose operation routines comprising a set of instructions for execution by said central processor unit in a prearranged manner to accomplish a specific task;

b. storing said set of general purpose operation routines in said read only memory;

c. defining a set of commands where each command of said set of commands is associated with a specific general purpose operation routine and including at least an operation code of a plurality of operation codes relating said command to said specific general purpose operation routine, each of said commands having an associated command code length which is either equal to or, in most cases, substantially less than the code length of said specific general purpose operation routine;

d. establishing at said remote computer system at least one application program module comprising a prearranged sequence of said commands;

e. communicating said application program module from said remote computer system to said local computer system via said communication channel, including the step of storing for each of said set of commands said operation code associated with said prearranged sequence of commands in said random access memory; and

f. establishing in said local computer system a program execution routine for enabling said central processor unit to execute said application program module by repetitively performing the steps of:

(1) reading said associated operation code stored in said random access memory;

(2) accessing a specific general purpose operation routine associated with said read operation code; and

(3) executing said specific general purpose operation routine.

2. In a method for operating a local terminal which includes a local computer system comprising a central processor unit, a read only memory coupled to said central processor unit, a random access memory coupled to said central processor unit, and a communication channel coupled to said central processor unit for communicating with a remote computer system, the steps of:

a. defining a set of different specific tasks to be carried out by said local computer system;

b. establishing a set of general purpose operation routines to be executed by said local computer system, each comprising a set of instructions for execution by said central processor unit in a prearranged manner to accomplish one of said specific tasks;

c. storing said set of general purpose operation routines in said read only memory;

d. defining a set of commands where each command of said set of commands is associated with a specific one of said general purpose operation routines and including at least an operation code relating said command to said specific general purpose operation routine, each of said commands having an associated command code length which is either equal to or, in most cases, substantially less than the code length of said associated general purpose operation routine;

e. establishing at said remote computer system at least one application program module comprising a prearranged sequence of said commands defining a desired functional feature for said local terminal;

f. communicating said application program module from said remote computer system to said local computer system via said communication channel, including the step of storing for each of said set of commands said operation code associated with said prearranged sequence of commands in said random access memory; and

g. establishing in said local computer system a program execution routine for enabling said central processor unit to execute said application program module by repetitively performing the steps of:

(1) reading said associated operation code stored in said random access memory;

(2) accessing a specific general purpose operation routine associated with said associated operation code; and

(3) executing said specific general purpose operation routine.

3. In a method for operating a local terminal to perform a set of predefined functions, said local terminal including a local computer system comprising a central processor unit, a read only memory coupled to said central processor unit, a random access memory coupled to said central processor unit, and a communication channel coupled to said central processor unit for communicating with a remote computer system, and input means for selecting one of said predefined functions, the steps of:

a. establishing a set of general purposes operation routines to be executed by said local computer system, each of said general purpose operation routines comprising a set of instructions for execution by said central processor unit in a prearranged manner to accomplish a specific task;

b. storing said set of general purpose operation routines in said read only memory;

c. defining a set of commands where each command of said set of commands is associated with a specific general purpose operation routine and including at least an operation code relating said commands to said specific general purpose operation routine, each of said commands having an associated command code length which is either to or, in most cases, substantially less than the code length of said specific general purpose operation routine;

d. establishing at said remote computer system a plurality of application programs, each of said application programs comprising a different set of application program modules which together define a different set of predefined functions for a local terminal to perform, each of said application program modules comprising a prearranged sequence of said set of commands to provide one of said predefined functions;

e. communicating a preselected application program from said remote computer system to said local

said program data structure array established in step h. includes a first slot which comprises a main slot into which data structures for a main application program module are loaded;

said DONE routine includes instructions for loading all bit locations in said ACTIVE PROGRAM bitmap with said second bit value indicating the absence of an active application program module except those bit locations corresponding to bit locations in said BACKGROUND bitmap having a first bit value indicating a background attribute for said active application program module; and

said step a. includes the step of:

(4) establishing as another one of said general purpose operation routines a START BACKGROUND routine comprising instructions for calling for execution of a specified application program module in a specified slot number with background attribute, including instructions to load the bit location in said ACTIVE PROGRAM bitmap with a first bit value indicating an active application program module in the specified slot number and to load the bit location in said BACKGROUND bitmap with a first bit value indicating a background attribute for said active application program module.

8. The method of claim 4, wherein

said local terminal includes at least one other input device coupled to said central processor unit in addition to said communication channel;

said scheduler data structure established in step i. further comprises a plurality of REALTIME PROGRAM data elements each associated with one of said program data structures in said program data array and storing in a specific bit location one of two data values indicating whether or not an active application program module in a corresponding slot of said program data structure array is waiting for input from one of said communication channel or said other input device;

said step a. includes the steps of:

(3) establishing a plurality of said general purpose operation routines include machine instructions utilizing inputs from said communication channel or said other input device; and

(4) establishing at least one general purpose operation routine which includes instructions for loading the bit location in said REALTIME PROGRAM data element corresponding to the application program module in which said general purpose operation routine is executed with a data value indicating that said application program module is waiting for input from said communication channel or said other input device;

and said scheduling program routine established in step j. enables said central processor unit to carry out the steps of:

(1) determining whether any input from said communication channel or said other input device is waiting to be read;

(2) when input is not waiting to be read, performing said step of scheduling the sequential execution of active application program modules one at a time based solely on the data values stored in said ACTIVE PROGRAM data elements in said schedule data structures, said execution of each active application program module continuing until said WAIT routine is executed or until said program module has completed execution;

(3) when input is waiting to be read, performing the steps of:

(a) reading an input from said communication channel or said input device; and

(b) scheduling the execution of only said active application program modules having a data value in a corresponding REALTIME PROGRAM data element which indicates that said application program module is waiting for input until said read input is used by one of said application program modules and then returning to said step (1) above.

9. The method of claim 8, wherein

said data structure format defined in step g. includes a plurality of DEVICE data elements each associated with said communication channel or said other input device and storing one of a first or a second data value indicating whether or not the specified application program module needs input from said communication channel or said other input device;

said step a. includes the step of:

(4) establishing as one of said general purpose operation routines a NEED INPUT routine which includes instructions for loading a specified set of said DEVICE data elements in an associated application program module with said first data value indicating that said application program module needs input from said communication channel or said other input device;

and said step (b) includes the steps of:

scheduling for execution one at a time in sequence each of said application program modules having a first data value in the corresponding REALTIME PROGRAM data element;

examining the DEVICE data element corresponding to the source of said input in said program data structure associated with each scheduled application program module to determine whether or not said scheduled application program module needs said read input;

if said DEVICE data element indicates that said read input is needed, proceeding to execute said scheduled application program module; and

if said DEVICE data element indicates that said read input is not needed, returning to step above to schedule the next appropriate application program module.

10. The method of claim 1, wherein

said step a. includes the steps of:

(1) establishing a subset of parameter dependent general purpose operation routines, each of said parameter dependent general purpose operation routines including instructions that use a predefined set of parameters, including a subset of parameters that are required to be supplied by an associated one of said set of commands, each of said parameters comprising one or both of a parameter address or a parameter value; and

(2) establishing for each of said parameter dependent general purpose operation routines a set of parsing routines for determining the address and value of each parameter in said subset of parameters;

said step c. includes the step of:

defining a subset of parameter dependent com-

mands, each of said parameter dependent commands being associated with one of said parameter dependent general purpose operation routines and including both a defined operation code and an ordered list of parameter types corresponding to said subset of parameters required to be supplied by said command;

said step d. includes the step of:

establishing for each use of said parameter dependent commands in said application program module an ordered list of actual parameter specifiers corresponding to said ordered list of parameter types;

said step e. includes the step of:

storing, in association with each of said set of commands of said operation code associated with a parameter dependent command, said ordered list of actual parameter specifiers;

said step f. (3) includes first performing with respect to each of said actual parameter specifiers the steps of:

(a) reading said parameter specifiers;

(b) executing an associated one of said parsing routines to obtain the address and value of said parameter;

(c) storing said parameter address and value;

followed by the step of executing said associated operation routine using one or both of said parameter address and value for each of said parameters.

11. The method of claim 1 where said general purpose operation routines have a specification of parameters, further comprising the steps of:

g-11. defining a set of data types that may be used to satisfy said specification of parameters utilized in said general purpose operation routines;

h-11. defining a set of parameter types that may be utilized in said general purpose operation routines, including defining for each of said parameter types an associated subset of said data types which may be used to satisfy said parameter types and a corresponding set of different data type values which may be utilized to specify said data types in said subset; one group of said parameter types being non-varying parameter types having an associated subset of only one of said data types, and another group of said parameter types being varying parameter types having an associated subset of a plurality of said data types;

i-11. establishing a set of data type parsing routines each comprising a set of instructions for obtaining at least the address and value of a parameter satisfied by an associated one of said data types;

j-11. establishing a set of parsing control routines each associated with one of said parameter types and comprising instructions for selectably executing one of a subset of said data type parsing routines corresponding to said associated subset of said data types, said subset having a single member when a parsing control routine is associated with one of said non-varying parameter types;

k-11. establishing a set of associated command parsing modules where each associated command parsing module is associated with at least one of said general purpose operation routines and comprising instructions for calling the execution of a subset of said parsing control routines in pre-established ordered sequence, said subset being a null subset in the case of each of said general purpose operation routines that do not utilize any of said parameter types;

l-11. storing each of said data type parsing routines, said parsing control routines, and said command parsing modules in said read only memory;

and wherein said step a. includes the steps of:

(1) establishing for each of said general purpose operation routines an operation routine which utilizes one or both of the address and value of a subset of said parameter types, said subset being null in the case of each operation routine that does not require any command specified parameters;

(2) incorporating into each of said general purpose operation routines a specific command parsing module; and

said step b. includes the step of:

storing said operation routine for each of said general purpose operation routines in prearranged relation to said specific command parsing module;

said step c. includes the step of:

defining for each of said commands a subset of said parameter types in an ordered list corresponding to one of said ordered sequences, said subset being a null subset when each command associated with an operation routine has an associated null subset of parameter types;

said step d. includes the step of:

establishing for each use of each of said commands in said application program module a subset of actual parameter type specifiers which are in the form of an ordered list and a set of data type data elements which are in the form of an ordered sequence of data elements corresponding to varying parameter types in said ordered list of actual parameter type specifiers, each of said data type data elements having a value designating which of said data types said actual parameter type specifiers are associated with;

said step e. includes the step of:

storing said ordered list and said ordered sequence established for each usage of each command in said random access memory in predetermined relation to said operation code;

said step f. (3) includes the steps of:

(a) first executing said associated command parsing module, including executing associated parsing control routines and data type parsing routines based on said ordered sequence to obtain the address and value of each parameter required by said operation routine; and

(b) storing the address and value of each of said parameters as they are obtained through execution of said data type parsing routines in prearranged locations in said random access memory; and

(c) then executing said operation routine associated with one of said general purpose operation routine utilizing one or both of said address and value of each of said parameters.

12. The method of claim 11, wherein said data type parsing routines, said parsing control routines, said command parsing modules, and said operation routines are stored in separate areas of said read only memory with each of said routines having a predetermined starting address;

said step c. comprises the steps of:

(1) assigning to each of said general purpose operation routines operation codes comprising a numerical value in sequential numerical order; and

(2) establishing an operation code jump table comprising a sequential listing of a first and a second address element for each of said operation codes in the same numerical order as said assigned order of operation code values, the first address element comprising the starting address of an associated one of said command parsing modules and the second address element comprising the starting address of an associated one of said operation routines;

said step f. (2) comprises looking up in said operation code jump table the first address element at the position in said jump table corresponding to said numerical value of said operation code; and

said step f. (3) (a) comprises executing the associated one of said command parsing modules located at said first address element looked up in said operation code jump table; and

said step f. (3) (c) comprises executing the one of said operation routines located at said second address element following said first address element.

13. The method of claim 12, wherein a plurality of said general purpose operation routines have a set of associated parameters comprising a null set; one of said command parsing modules comprises a null command parsing module which calls a null set of parsing control routines and returns to carry out step f. (3) (c).

14. The system of claim 1, wherein said step b. comprises storing each of said general purpose operation routines at a prearranged location within said read only memory having an associated starting memory address;

said step c. comprises the steps of:

(1) assigning to each of said general purpose operation routines an operation code comprising a numerical value in sequential numerical order; and

(2) establishing an operation code jump table comprising a sequential listing of said starting memory address for each of said operation codes; and wherein said step f.(2) comprises looking up in said operation code jump table the starting memory address of said associated general purpose operation routine at the listing in said jump t able corresponding to said numerical value of said operation code and said step f.(3) comprises the step of sequentially executing the instructions stored in said read only memory beginning at said starting memory address.

15. The system of claim 1, further comprising the step of:

g-15. defining an operation code system comprising a plurality of opcode sets and a plurality of corresponding opcodes within each of said sets, such that each operation code comprises an opcode set designation and an opcode designation;

h-15. defining an object code command syntax comprising at least one data element having a set of possible values each designating one of said opcode sets and at least a second data element having a set of possible values each designating one of said operation codes;

and wherein said step c. includes the steps of:

(1) assigning each of said commands to a specific one of said opcode sets and assigning to each of said commands in each opcode set a different opcode in sequential numerical order;

(2) establishing a set of associated opcode jump tables each corresponding to one of said opcode sets and comprising a sequential listing of operation routine pointers each comprising starting memory addresses for the general purpose operation routines associated with the commands assigned to said corresponding opcodes;

(3) storing said set of opcode jump tables in said read only memory at prearranged starting addresses for each of said tables in said set;

(4) establishing an opset pointer table comprising a sequential listing of said starting memory addresses for said associated opcode jump tables; and

(5) storing said opset pointer table in said read only memory at a prearranged starting address;

said step e. includes the step of:

storing said one data element and said second data element in associated locations in said random access memory;

said step f. includes establishing a command execution data structure including an opset data structure element and an opcode data structure element; said step f.(1) includes the steps of:

(a) reading said one data element and storing the read value in a opset data structure element; and

(b) reading said second data element and storing the read value in said opcode data structure element;

and said step f.(2) includes the steps of:

(a) calculating the pointer address of an opset pointer from said starting memory addresses of said opset pointer table and said value stored in said opset data structure element;

(b) reading the value stored at said pointer address;

(c) calculating the opcode pointer address of said opcode pointer from said read value and said value stored in said opcode data structure element; and

(d) reading the value stored at said opcode pointer address:

and said step f.(3) comprises the step of:

executing the general purpose operation routines stored in read only memory at the addresses corresponding to the read value at said pointer address.

16. The method of claim 1 wherein said general purpose operation routines have a specification of parameters, further comprising the steps of:

g-16. defining a set of data types that may be used to satisfy said specification of parameters utilized in said general purpose operation routines;

h-16. defining a set of parameter types that may be utilized in said general purpose operation routines, including defining for each of said parameter types an associated subset of said data types which may be used to satisfy said parameter types and a corresponding set of different data type values which may be utilized to specify said data types in said subset; one group of said parameter types being non-varying parameter types having an associated subset of only one of said data types; and another group of said parameter types being varying parameter types having an associated subset of a plurality of said data types;

i-16. establishing a set of data type parsing routines each comprising a set of instructions for obtaining the address and value of a parameter satisfied by an associated one of said data types;

j-16. establishing a set of parsing control routines each associated with one of said parameter types and comprising instructions for selectably executing one of a subset of said data type parsing routines corresponding to said associated subset of said

data types, said subset having a single member when a parsing control routine is associated with one of said non-varying parameter types;

k-16. establishing a set of command parsing modules each associated with at least one of said general purpose operation routines and comprising instructions for executing of a subset of said parsing control routines in pre-established ordered sequences, said subset being a null subset when each of said general purpose operation routines do not utilize any of said parameter types;

l-16. storing each of said data type parsing routines, said parsing control routines, and said command parsing modules in said read only memory;

g-16. defining an operation code system comprising a plurality of opcode sets and a plurality of opcodes within each of said sets, such that each operation code comprises an opcode set designation and an opcode designation;

h-16. defining a object code command syntax comprising a set of bitmap bytes, a set of opcode bytes, and a set of parameter type specifier bytes, each of said bitmap bytes and said opcode bytes having a set of common bit positions comprising a byte-type flag and storing a first numeric value in each bitmap byte and a second numeric value in each opcode byte, each of said bitmap bytes also having a set of common bit positions comprising opset specifier data elements and a plurality of sets of common bit positions comprising data type data elements; each of said opcode bytes having a set of common bit positions comprising opcode specifier data elements; said opset specifier data elements in said set of bitmap bytes together identifying the opcode set of the command as a particular one of said opcode sets, said data type data elements in said set of bitmap bytes together identifying the data type values for all varying parameter types in the command that do not comprise the default data type value; said opcode specifier data elements in said set of opcode bytes together identifying the opcode of the command as one of said opcodes;

and wherein said step a. includes the steps of:

(1) establishing for each of said general purpose operation routines an operation routine which utilizes a subset of said parameter types, said subset being null for each operation routine that does not requires any command specified parameters;

(2) incorporating into each of said general purpose operation routines one of said command parsing modules; and

said step b. includes the step of:

storing said operation routine for each of said general purpose operation routines in prearranged relation to said command parsing modules associated therewith;

said step c. includes the step of:

(1) defining for each of said commands a subset of said parameter types in an ordered list corresponding to one of said ordered sequences of parsing control routines in one of said command parsing modules, said subset being a null subset when each command associated with an operation routine has an associated null subset of parameter types;

(2) assigning each of said commands to a specific one of said opcode sets and assigning to each said command in each opcode set a different opcode in sequential numerical order;

(3) establishing a set of opcode jump tables each corresponding to one of said opcode sets and comprising a sequential listing of operation routine pointers each comprising starting memory addresses for the one of said general purpose operation routines associated with the command assigned to said corresponding opcode;

(4) storing said set of opcode jump tables in said read only memory at prearranged starting addresses for each of said tables in said set;

(5) establishing an opset pointer table comprising a sequential listing of said starting memory addresses for said opcode jump tables; and

(6) storing said opset pointer table in said read only memory at a prearranged starting address;

said step d. includes the step of:

establishing for each use of each of said commands in said application program module object code commands in accordance with said object code command syntax;

said step e. includes the step of:

storing each of said object code commands in said application program module in said random access memory;

said step f. includes establishing a command execution data structure including an opset data structure element, an opcode data structure element, an array of data type data structure elements, an array of parameter address and value data structure elements, a variable parameter tracking data element, and a current parameter tracking data element;

said step f.(1) includes the step of:

decoding said set of bitmap bytes and said set of opcode bytes to obtain an opset value and store said opset value in said opset data structure element, to obtain an opcode value and store said opcode value in said opcode data structure element, and to obtain data type values and store data type values of all of said data type data elements in individual ones of said data type data structure elements;

and said step f.(2) includes the steps of:

(a) calculating the address of an opset pointer from said starting address of said opset pointer table and said value stored in said opset data structure element;

(b) reading the value stored at said opset pointer address;

(c) calculating the address of said opcode pointer from said read opset pointer value and said value stored in said opcode data structure element; and

(d) reading the value stored at said opcode pointer address as the address of the one of said gengral purpose operation routines associated with said command;

said step f.(3) includes the steps of:

(a) first executing said command parsing module associated with said general purpose operation routine at said address read in step f.(2) (d) above, including executing associated parsing control routines and data type parsing routines based on said data type values stored in said data type data structure elements to read the parameter types in said object code command and to obtain the address and value of each parameter required by said operation routine; and

(b) storing the address and value of each of said parameters as they are obtained through execution of

said data type parsing routines in individual ones of said parameter and address data structure elements;

(c) then executing said operation routines associated with said general purpose operation routines utilizing one or both of said address and value of each of said parameters.

17. In a method for programming a local terminal, the steps of:

establishing a tokenized interpretive programming language having a plurality of commands and corresponding syntax including a source level command and parameter syntax, an associated object level command and parameter syntax including a defined operation code for each of said commands, a program module structure, application modules and a compiler for translating source level commands in said application modules to said object level command capable of being assembled and linked into binary code;

establishing a set of operation routines for execution by said terminal to perform a set of terminal tasks;

storing said operation routines in said local terminal;

defining a set of commands using said source level command and parameter syntax, each being associated with one of said operation routines;

creating an application program comprising a sequence of said source level command in accordance with said program module structure;

compiling said application program using said compiler;

assembling and linking said program using a standard machine code assembler and linker program;

communicating said program to said terminal for storage in local memory therein; and

operating said terminal using said application program.

18. The method of claim 17, wherein said step of compiling comprises the steps of establishing a first section of compiled code comprising a fixed code structure that defines the environment of the application program and a variable code structure that comprises portions of the application program that vary with the content of the program created in said creating step whereby said

assembling and linking step creates a corresponding fixed binary code section and a variable binary code section;

and said step of communicating said application program comprises first communicating said fixed binary code section and storing it at the local terminal and then communicating said variable binary code section and storing it in the local terminal;

said fixed binary code section stored in said local terminal providing information for accessing portions of said application program in said variable binary code section stored in said local terminal while performing said step of operating said terminal using said application program.

* * * * *

METHOD FOR OPERATING A LOCAL TERMINAL TO EXECUTE A DOWNLOADED APPLICATION PROGRAM

Field of the Invention

Background and Prior Art

1. Demand for Point of Transaction Terminals with Custom Application Programs
2. Prior Art Approaches to Custom Terminal Programming
3. Prior Art of Downloading Program Information

Summary of the Invention

1. Objects of this Invention
2. Features and Advantages of this Invention
   - a. Reduced Code Size for Application Program Download
   - b. The Programming Language Feature of This Invention
   - c. Concurrent Scheduling of Application Program Modules

Brief Description of Drawing Figures

Detailed Description of Embodiments of the Invention

BACKGROUND ENVIRONMENT OF THE INVENTION

1. Illustrative Example of Local Terminal Features and Operation (FIGS. 1A and 1B)
   - a. The Telephone Function
   - b. The Calculator Function
   - c. The Transaction Terminal Functions
   - d. Standard and Custom Terminal Features
2. Illustrative Example of Local Terminal Hardware Environment for the Invention (FIGS. 2 and 3)

GENERAL METHOD OF THIS INVENTION

1. The Core Programming Methodology
   - a. Establishing the General Purpose Operation Routines
   - b. Defining a Set of Commands
     - (1) General Aspects
     - (2) Preferred Command Structure and Syntax
     - (3) Object Code Syntax of Defined Commands (FIGS. 4A and 4B)
     - (4) Alternative Operation Code Systems and Associated Object Code Command Syntax (FIGS. 5-8)
     - (5) Compiling Source Code Commands into Object Code Command Syntax
   - c. Storing the General Purpose Operation Routines
     - (1) The Preferred Storage and Accessing System (FIG. 9)
     - (2) Alternative Storage and Accessing Systems (FIGS. 10 and 11)
2. Establishing an Application Program Module
3. Communicating the Application Program Module to the Local Terminal
4. Establishing a Program Execution Routine
   - a. Data Structures and Arrays used in Executing Application Program Module (APM) Commands (FIGS. 12 and 13)
   - b. Execution of an Application Program Module Command (FIG. 14)
     - (1) Decoding Bitmap and Opcode Bytes
     - (2) Checking Validity of Opset and Opcode Values
     - (3) Parsing Command Parameters and Executing the Operation Routine
   - c. Specific Example of Execution of an Application Program Module Command
     - (1) Executing Command Parsing Module XA (FIGS. 9 and 15)
       - (a) Parsing Control Routine for Parameter Type X (FIG. 16)
       - (b) Parsing Routine BC for Constant-Byte Data Type (FIG. 17)
       - (c) Parsing Routine VS for Variable-Byte Data Type (FIG. 18)
       - (d) Parsing Routine EB for Extended-Byte Data Type (FIG. 19)
       - (e) Parsing Routine SV for Stack Ref.-Byte Data Type (FIG. 20)
       - (f) Parsing Control Routine for Parameter Type A (FIG. 21)
       - (g) Parsing Routine EA for Extended-Address Data Type (FIG. 22)
       - (h) Other Data Type Parsing Routines
     - (2) Executing Associated Operation Routine—The Start Job Command (FIGS. 23-25)
       - (a) The APMACTIVE Routine (FIG. 24)
       - (b) The STARTAPM Routine (FIG. 25)
   - d. Other Application Program Modules Utilizing the XA Type of Parsing Control Routine
     - (1) The CHK_JOB_DONE (CHK.APM.-DONE) Command (FIG. 26)
     - (2) The START_BACK Command
   - e. Other APM Commands Which Use Both Parameter Address and Value—The INC_INT Command (FIG. 52)
5. Alternative Command Execution Routines
   - a. Alternative Routines Involving Alternative APM Storage and Accessing Systems
   - b. Alternative Routines for Executing APM Commands Using Alternative Bitmap/Operation Code Systems (FIGS. 27 and 28)

ALTERNATE IMPLEMENTATION OF GENERAL METHOD OF THE INVENTION

1. State Table Module
2. Interpreter Routine
   - a. Entry from Ready State
   - b. Opcode (APM Command) Execution Routine
   - c. Opcode (APM Command) Termination
3. Opcode Group of Operation Routines
4. External Event Processing Routines

APPLICATION PROGRAM MODULE SCHEDULING METHOD

1. Data Structures Associated with the Scheduler Methodology (FIG. 12)
   - a. The Bitmaps of the Scheduler Data Structure (FIG. 12A)
   - b. Other Data Elements in the Scheduler Data Structure
2. One Version of the Scheduler Methodology of this Invention (FIG. 29)
   - a. The Read Input Routine (FIG. 30)
   - b. The ACTION.START.APM Routine (FIGS. 31 and 32)
   - c. The ACTIVE SCHEDULER Routine (FIG. 33)
   - d. The CHECK/EXECUTE Routine (FIG. 34) as Called by the ACTIVE SCHEDULER Routine
   - e. The REALTIME SCHEDULER Routine (FIG. 35)
   - f. The CHECK/EXECUTE Routine (FIG. 34) when called by the REALTIME SCHEDULER Routine
   - g. Alternative CHECK/EXECUTE Routines (FIG. 36)

h. Initiating APM Execution from Input
(1) State Table Look-Up of Input—the STATE.-TABLE.LOOKUP Routine (FIGS. 37A and 37B)
(2) Alternative Approaches to Initiating Execution of APMs
i. Execution of the DO.ACTION Routine (FIG. 52)
j. Execution of APM Commands which Terminate Execution of APMs
(1) The HALT Command (FIGS. 38-40) and the ABORT.STATE.TABLE.ACTIONS Routine (FIG. 49)
(2) Executing the STOPAPM Routine from the HALT Routine (FIG. 39)
(3) The SLOT.KILL Routine (FIG. 40)
(4) The STOPAPM Routine Continued (FIG. 39)
(5) The ABORT.STATE.TABLE.ACTIONS Subroutine (FIG. 49)
(6) The R_ABORT Action and Associated Subroutines (FIGS. 53, 49, and 54)
(7) The DONE Command and Associated Operation Routine (FIG. 41)
(8) The STOPAPM Routine as Executed from the DONE Routine (FIGS. 39 and 41)
(9) Executing the STOP_JOB Command and Associated Operation Routine (FIG. 42)
(10) The STOP_BACK Command and Associated Operation Routine (FIG. 43)
k. Commands and Associated Operation Routines that Suspend Execution of APMs
(1) The WAITFOR_JOB Command and Associated Operation Routine (FIG. 44)
(2) The NEED_INPUT, CHK_INSRCE and WAIT_JUMP Command Sequence
(a) The NEED_INPUT Command and Associated Operation Routine (FIG. 45)
(b) The CHK_INSRCE Command and Associated Operation Routine (FIG. 46)
(c) The WAIT_JUMP Command and Associated Operation Routine (FIG. 47)
l. Execution of other APM Commands which Manipulate Bitmaps in the Scheduler Data Structure
3. Example of an APM which Utilizes Concurrent Execution of Subsidiary APMs
4. Alternative Versions of Scheduler Methodology
OTHER PROGRAM COMPONENTS RESIDENT IN THE LOCAL TERMINAL
1. The Power-On Routine (FIG. 49)
2. The Executive Error Routine (FIG. 50)
3. The Download Control Code
4. Other Miscellaneous Program Components
a. Data Entry Routines
b. Control String Interpreter
c. The Operating System
d. File Manager System
e. Direct Download Program
f. Miscellaneous Support Routines
ESTABLISHING AN APPLICATION PROGRAM
1. The Tools Utilized in Establishing an Application Program
a. The Preprocessor Program: PREZAPD
b. The Assembler Program: ZAPDASM
2. Creating the Source Code of the Application Program
PERFORMING THE PROGRAM COMMUNICATION STEP
1. General Aspects of the Communication Step
2. The ROM-resident Routine R_LOADON Executed to Call For Download of an Application Program (FIG. 55)
3. The Communication Protocol in a Specific Version of the Application Program Downloading Methodology
4. The Downloaded Data in a Specific Version of the Communication Step of This Invention
a. General Aspects
b. The Binary Download
(1) The Fixed Portion
(2) The Variable Size Portion
c. The File and Record Download
ADVANTAGES OF THE METHODOLOGY OF THE INVENTION

FIELD OF THE INVENTION

This invention relates generally to methods for operating a local terminal which includes a programmable computer so that the terminal executes a prearranged application program. More specifically, this invention relates to methods for operating a local terminal according to a prearranged application program which is created on a remote computer, then communicated via a communication channel to the local terminal where it is stored for execution.

BACKGROUND AND PRIOR ART

1. Demand for Point of Transaction Terminals with Custom Application Programs

Substantial performance improvements in integrated circuits for the microcomputer and telecommunications fields have been made in recent years so that functionally enhanced integrated circuits are available at the same or, lower price. These cost/performance improvements have made it attractive to mass produce and deploy computer based systems which employ integrated circuit technology. One such system application is the point of sale transaction terminal for credit verification and credit transaction data capture. Another application is the automated bank teller terminal which enables the unattended performance of a number of types of banking transactions.

Other applications involve a wide variety of local data capture terminals which communicate with a remote host computer. It is anticipated that such system applications will eventually extend the technology into home banking terminals for transfer of money between accounts, automated payment of bills, and potentially a wide variety of other financial and non-financial transactions.

The principal customers for point of sale transaction terminals of all types are major financial institutions, including banks, savings and loan associations, and credit operations of major chains of retail stores. For a variety of reasons, different customers for point of sale transaction terminals need or desire to customize the functions of their terminals. Banks and other financial institutions are in the process of extending their walls to the merchant's checkout counter and point of sale transaction terminals are a major factor in this extension of services.

For strong merchant acceptance, the features of point of sale transaction terminals must be tailored to meet the

specifically perceived, different needs of each of the different merchant groups that represent the ultimate customers for the terminal. Even among merchant groups where the services needed are essentially the same, there is no standard approach to providing these services. Thus to satisfy the needs and demands of the marketplace, point of sale transaction terminals must be able to be customized for the particular application at which they are directed. To achieve this result, the computer program being executed by the local computer system in the local terminal must be a custom application program. In addition to providing point of sale transaction terminals initially with custom application programs, customers will often need or desire to alter the application program to add or change a feature or to remedy a problem or defect discovered in the program after the terminal is installed. The customer desiring or needing a program revision may have a large number of terminals physically located at many different places.

2. Prior Art Approaches to Custom Terminal Programming

Providing custom application programs for point of sale transaction terminals results in major problems for both the vendor of the terminals and the customer. The standard approach to programming a local terminal is to load the application program into one or more integrated circuits of the programmable read only memory type and then to physically install those program memory circuits in the terminals to be supplied to the customer. If the vendor is supplying terminals with different custom application programs to a number of different customers, there is a basic problem associated with maintaining segregated inventories of the different program memory circuits.

In a typical case of generating a custom application program for a particular customer, the program will evolve through several versions before it achieves satisfactory functionality. When the program is accepted by the customer, typically thousands of terminals are delivered and installed at widely separated geographic locations. Even with exercise of great care in developing and testing the program before delivering large quantities of terminals, it is very likely that certain minor program deficiencies will be detected by users of the terminals during the first few weeks and months of operation.

If the application program is stored in the local terminal in programmed read only memory, these program deficiencies can only be remedied by revising the application program, loading it into new memory circuits, and then installing the new memory circuits in each of the terminals by visiting each installation site or by bringing the terminals back to a central facility for installation of the new program memory circuit. Thus each program change causes an expensive and time-consuming logistics problem with all the attendant disruptions of the business of the customer.

The problems encountered with remedying program deficiencies are also present when the vendor and customer contemplate a program change to add or change features and wish to modify some or all of the terminals that are already installed in the field. The logistical problems tend to discourage enhancing the custom programs being executed in the terminals after satisfactory operation of the initial program has been achieved. Yet it is often very important to the customer to be able to enhance the program to meet competition or to keep up with the growing needs or expectations of the merchants or other parties that are using the terminals in their day to day business operations.

3. Prior Art of Downloading Program Information

The point of sale transaction terminals of the type to which this invention is directed are invariably installed with a communication link to a host computer. This has lead to consideration of implementing one solution to the logistical problem of updating existing terminals in the field by communicating the new program to the individual terminals in the field via the communication link with the host computer.

It has been known in the art for some time that it is feasible to download limited types of control information to a local terminal to alter the manner in which a relatively fixed local application program is executed. For example, there are systems in which the prompts presented to the operator of the terminal may be downloaded from the host to the local terminal. This may be done in a static sense, with the downloaded information stored in memory at the local terminal until it is changed by the host. Alternatively, it may be done in a dynamic mode, with the host controlling the prompts in an online manner.

It has also been known in the prior art to download from the host computer to the local terminal certain file and control information to be used by an otherwise fixed program in the terminal. For example, it is known to download such information as the telephone number for the terminal to dial or log-on information for accessing a remote network. It is also known to download a control string which controls the building of a data packet for transmission from the local terminal to the host computer. Control string downloads have also been used to control the files in which responses from the terminal are stored.

There have also been limited implementations in the prior art of downloading of entire machine language application programs from a remote computer to a local terminal with storage of the downloaded program in battery protected random access memory circuits at the local terminal. In this case the read only memory circuits at the local terminal contain only the programs for basic management of the download of the application program together with the operating system for the basic program control of the central processor unit in the terminal.

While, on the surface, this approach to customizing the application programs running on point of sale transaction terminals may look attractive, there are several reasons why a complete machine language program download is impracticable. The principal reason is the size of the machine language code and thus the time required to perform the download for each local terminal. It is not practicable to author the custom application program directly in machine language because of the scarcity of programmers who can author programs directly in machine language. Consequently it is necessary that the program be initially written in a high level language such as the C-language or Pascal, and then compiled and linked into directly executable machine code. The compiler and linker inherently produce a machine language version which is substantially larger in overall code size compared to the code size for an equivalent program written by a creative programmer directly in machine language.

Reasons for a compiler/linker

program size

This larger code size is not a problem when the code is directly loaded into read only memory program circuits for physical insertion into the local terminal because automated programming systems are available to load the code into a number of read only memory circuits simultaneously. The larger code size becomes a problem only when direct downloading of the entire code into the local terminal is contemplated.

To illustrate, a compiled and linked version of a typical application program may comprise 32 kilobytes of code. Most local terminals communicate with the host computer via a relatively inexpensive modem running at a data transmission rate of 300 baud. At that transmission rate, download of the application program to the local terminal will take about 20 minutes. This might be acceptable if there were only a few terminals to download from the remote host computer. In the typical case, it may be necessary to download several thousand local terminals. Assuming a download time of 20 minutes and 3000 terminals to download, a total of 60,000 minutes or 1000 hours of downloading time would be required. If the downloading were done by a single host computer during the twelve hours between 8 pm and 8 am, it would take over 80 days to download all of the terminals. It is thus apparent that the concept of downloading a complete machine language application program is impracticable where a large number of terminals are involved.

SUMMARY OF THE INVENTION

1. Objects of this Invention

It is the principal object of this invention to provide an improved method of operating a local terminal to execute a custom application program.

It is another object of this invention to provide a practicable method of operating a local terminal based on a custom application program downloaded from a remote host terminal.

2. Features and Advantages of this Invention

a. Reduced Code Size for Application Program Download

This invention features a method for operating a local terminal which includes a computer system comprising a central processor unit, a read only memory coupled to the central processor unit, a random access memory coupled to the central processor unit, and a communication channel coupled to the central processor unit for communicating with a remote computer system. Other input devices, such as keyboards, PIN code entry pads, cardreaders, and the like may also be provided in the local terminal. The method of this invention provides a practicable approach to downloading an entire custom application program because the size of the program code required to be downloaded is reduced by a factor of three or four by avoiding the requirement of downloading all of the actual machine code instructions of the application program.

Instead, the method of this invention is based on storing in read only memory circuits within the local terminal a number of general purpose operation routines which comprise instructions to be executed by the central processor unit to accomplish a particular program task. Each of these general purpose operation routines is associated with a defined command which, in its object code version, includes an operation code. The object code version of the commands associated with the general purpose operation routines has a code length substantially shorter than the code length of the operation routine. In accordance with the method of this invention, the local terminal also has a program established in its read only memory system for interpreting the operation code to access the associated general purpose operation routine for execution by the central processor unit. The object code version of the application program in the form of a sequence of commands has a code size which is several times smaller than the compiled code size would be for an entire application program which could be directly executed after downloading.

More specifically, the method of this invention includes a first step of establishing a set of general purpose operation routines to be executed by the local computer system. Each of these general purpose operation routines comprises a set of instructions for execution by the central processor unit in a prearranged manner to accomplish a specific task. The next step is to store the set of general purpose operation routines in the read only memory of the local terminal so that they may be accessed for execution by the central processor unit. The storage locations of these general purpose operation routines will be arranged and noted so that the routines can be addressed.

A following step is to define a set of commands, each of which is associated with a specific one of the general purpose operation routines. Each of the commands includes at least an operation code relating the command to its associated general purpose operation routine. Each of the commands is defined such that it has an associated command code length substantially less than the code length of the associated general purpose operation routine. Preferably, for convenience of writing programs, each command is defined with a high level programming syntax in which the command is represented by a series of word forms in abbreviated notation which have a recognizable association with the task that the associated general purpose operation routine will perform when it is executed by the central processor unit in the local terminal. This high level form of the command is then compiled and assembled to produce the operation code which relates to the associated general purpose operation routine in a manner which is intelligible to the interpreter program in the local terminal.

Once these foregoing steps have been performed, then next step in the method of this invention is to establish at the remote computer system at least one application program module comprising a prearranged sequence of the commands which have been defined. This application program module is written such that, when the associated general purpose operation routines are executed by the central processor unit in the local terminal, a meaningful series of tasks associated with the desired functionality of the local terminal will be carried out. Typically, a custom application program running in a point of sale transaction terminal will utilize a multiplicity of application program modules which will be called for execution in a particular sequence to carry out a complete transaction. The simplest and broadest form of the invention, however, contemplates that there may be one or more application program modules.

The next step in the method of this invention is to communicate the application program module from the remote computer system to the local computer system via the communication channel. This includes the step

of storing the operation codes associated with the prearranged sequence of commands in the random access memory of the local terminal.

The method further includes the step of establishing in the local computer system a program execution routine for enabling the central processor unit to execute the application program module. This is accomplished by repetitively performing the steps of:

(1) reading one of the operation codes stored in the random access memory;

(2) accessing a specific one of the general purpose operation routines associated with the read operation code; and

(3) executing the accessed general purpose operation routine.

It will be appreciated that the core methodology of this invention provides a number of advantages which makes it practicable to download custom application programs in terms of the time required for the downloading. As will be better understood from the detailed description given below, the method of this invention involves a constructive and feasible tradeoff of program execution speed at the local terminal for spped of downloading the custom application program.

The speed of execution of the application program by the local terminal is reduced because of the steps involved in "interpreting" the operation code to access the associated general purpose operation routine so that it can be executed by the central processor unit. This interpreting step is not present when a machine language program stored in the local terminal is directly executed. It has been found, however, that this reduction of speed of execution at the local terminal is not a problem provided that the program execution overhead which is occupied by the interpreter program is kept at a manageable level.

Furthermore, the tradeoff in overall speed of execution can be reduced by creative authoring of the general purpose operation routines as highly efficient machine language routines. Since this authoring is done only once, it is feasible to have a very skilled machine language programmer spend a substantial amount of time to optimize the code of each routine. The set of routines can also be optimized to the type of transactions which are performed by the local terminal. When this is done, some of the time that is lost in accessing the general purpose operation routine to be executed can be made up in speed of execution of the routine itself.

More importantly, it has been found that, with creative implementation of the core concepts of this invention, it is possible to reduce the download time for a custom application program by a factor of two to three. A program which might otherwise take 20 minutes to download, can be downloaded in about 7-10 minutes using the method of this invention. This dramatic reduction in download time makes it feasible to download a new version of a custom program into a large number of local terminals within a reasonable time frame.

The method of this invention thus avoids all of the logistic problems associated with the prior art approach of loading the custom program into read only memory circuits physically installed in each of the local terminals. Instead, the local terminal is provided with read only memory for storing the operating system and the general purpose operation routines as well as the other locally resident programs modules and facilities which are required to implement the method. These read only memory circuits are common to all of the local terminals being sold regardless of the application program which the terminal will run. Random access memory is provided for storing the application program as well as alterable parameter and file information. Thus a terminal can be completely manufactured and inventoried for sale to any one of several ultimate customers having differing application program requirements. Prior to delivery the standard terminal can be downloaded with a standard application program or a custom program provided for that customer alone.

In initial programming of the local terminal, the download of the application program into random access memory terminal can be accomplished by a high speed direct data link, such as through the RS232 port communicating with a previously downloaded terminal at 9600 baud. This enables initial programming of the additional local terminals within less than one minute each, typically. Then after the terminals are installed and operating in the field, they can be conveniently reprogrammed to eliminate program deficiencies or to add enhanced features by downloading via the telephone lines through the modems at the host computer and the local terminal.

In addition to the core programming and program execution feature of this invention and its advantages as part of a method of operating a local terminal, this invention includes a number of other, related advantageous features:

(1) the methodology in which operation codes are grouped in plural groups and are defined at the object code level using a bitmap/opcode system.

(2) the methodology for handling specification of parameters within the defined commands and indicating the data types of varying data type parameters using data elements integrated into the bitmap/opcode system.

(3) the methodology for establishing general purpose operation routines which share program facilities within the local terminal for parsing the parameters which are included within the defined commands.

(4) the methodology of integrating parameter parsing routines and control of the selection of parameter parsing routines for varying parameters satisfied by different data types using data type data elements included in a bitmap/opcode system in the object code command syntax.

b. The Programming Language Feature of This Invention

This invention also features a method for programming a local terminal which involves the key step of establishing a tokenized interpretive programming language system having a source level command and parameter syntax, an associated object level command and parameter syntax including a defined operation code for each of the commands, a program module structure, and a compiler for translating source level commands in the application modules to the object level commands capable of being assembled and linked into binary code.

Compiler
linker
CLS

This programming method also involves establishing a set of operation routines for execution by the terminal to perform a set of terminal tasks. These operation routines are stored in the local terminal. An additional step in this method is defining a set of commands using the source level command and parameter syntax, each being associated with one of the defined operation routines.

After these steps have been performed the next step is creating an application program comprising the source level sequences of the defined commands in accordance with the program module structure. This is followed by compiling the application program using the compiler, and then assembling and linking the compiled program using a standard machine code assembler and linker program. The assembled and linked program is then communicated to the terminal for storage in local memory therein; and then the terminal is operated using the stored application program.

This methodology provides the capability of achieving higher programming productivity because of the high level nature of the commands. The commands can utilize command names which are self-documenting for the task that the associated general purpose operation routines will perform at the local terminal. The use of operation codes at the object code level of the command syntax provides the compressed download code feature of this invention.

c. Concurrent Scheduling of Application Program Modules

A second principal feature of this invention involves providing a methodology for psuedoconcurrent execution of application program modules. This methodology involves providing defined commands which can be incorporated within one application program module to start the execution of another application program module concurrently or to cause one of the active application program modules to wait or suspend its execution in favor of going forward with execution of a different active application program module, and by establishing a scheduler program to schedule the execution of the active application program modules so that a meaningful and organized approach to starting and stopping execution of individual active application program modules is attained. The preferred scheduler program feature includes methods for giving priority to active application program modules that need input from input devices or a timer.

This feature of this invention provides the advantage of greater flexibility in the structuring of application program modules to maximize the use of the program execution capability of the microprocessor within the local terminal. The scheduler program facility and the associated features of the defined commands that are integrated with the scheduler do not provide the capability for two application program modules to be actually running simultaneously. However, the scheduler and these commands enable the programmer to interleave the different application program modules that are active in the local terminal at any one time period in such a way that it appears that a number of tasks are being carried out at the same time. At places in the execution of a particular application program module where it cannot continue to execute without waiting for something to happen, such as waiting for a particular input to be received, or waiting for another application program module to finish its execution, the current application program module may suspend its execution with a WAIT command, so that other application program modules can proceed with their execution under the control of the scheduler.

From the above summary of the features of this invention, it should be apparent that this invention provides the capability of structuring operation routines and commands associated therewith such that the application program development tools facilitated by this invention can be optimized for the types of transactions being carried out by the local terminal. Furthermore, the overall methodology of this invention makes it possible to create a high level, pseudo-concurrent program language which facilitates the development of custom application programs for point of sale transaction terminals. This invention solves the difficult problem of manageable application program download times. It thus encourages users of point of sale transaction terminals to incorporate features and functions which maximize the utility and value of the local terminals in the business environment in which they are deployed.

Other objects, features and advantages of this invention will be apparent from a consideration of the detailed description given below in conjuction with the accompanying drawings.

BRIEF DESCRIPTION OF DRAWING FIGURES

FIGS. 1A and 1B are illustrations of a local terminal in which the method of this invention may be practiced.

FIGS. 2 and 3 are block schematic diagrams of the circuitry of a local terminal in which the method of this invention may be practiced.

FIGS. 4A and 4B illustrate an operation bitmap and opcode system useful in practicing the method of this invention.

FIGS. 5A, 5B, 6, 7 and 8 illustrate alternative version of bitmap and opcode systems that may be used in practicing the method of this invention.

FIG. 9 illustrates a preferred system for storing and accessing operation routines and parsing routines in accordance with this invention.

FIGS. 10 and 11 illustrate alternative systems for storing and accessing operation routines and parsing routines in accordance with this invention.

FIGS. 12A-12E illustrate data structures and data element bit assignments useful in practicing the method of this invention.

FIGS. 13A and 13B illustrate data structure and data storage arrays useful in the parameter parsing methods of this invention.

FIGS. 14-22 are flowcharts illustrating routines associated with executing application program module commands in accordance with this invention.

FIGS. 23-26 are flowcharts illustrating operation routines associated with application program module commands provided in accordance with this invention.

FIGS. 27 and 28 are flowcharts illustrating alternative routines for executing application program module commands in accordance with this invention.

FIGS. 29-31 are flowcharts illustrating routines useful in implemnting the concurrent application program module scheduling methods of this invention.

FIG. 32 illustrates a system for storing and accessing application program modules in accordance with this invention.

FIGS. 33-35 are flowcharts illustrating routines useful in implementing the concurrent application program module scheduling methods of this invention.

FIGS. 36A and 36B illustrate an alternative routine and data element bit assignment useful in implementing the concurrent application program module scheduling methods of this invention.

FIG. 37A illustrates a compressed state table storage and accessing system useful with this invention.

FIG. 37B illustrates a state table lookup routine useful with this invention.

FIGS. 38–47 are flowcharts illustrating routines associated with application program module commands useful in implementing this invention.

FIGS. 48–55 are flowcharts of routines which are useful in connection with implementing this invention.

DESCRIPTION OF EMBODIMENTS OF THE INVENTION

BACKGROUND ENVIRONMENT OF THE INVENTION

This invention involves a method for operating a local terminal to execute a prearranged application program which has been downloaded into the local terminal from another computer system. Execution of the application program enables the local terminal to perform a predefined set of functions. These functions are specific to the particular application to which the local terminal is dedicated, but may be altered by downloading a revised application program from a remote computer.

1. Illustrative Example of Local Terminal Features and Operation (FIGS. 1A and 1B)

The invention will be illustrated by discussing its use in a specific local terminal application, namely a point of sale transaction terminal of the type which is purchased by a retail merchant through a credit provider such as a local bank for the purpose of carrying out credit transactions with customers and related data capture functions. It should be understood, however, that the invention is not limited to this type of local terminal application, but could be used in a number of other local terminal applications having similar requirements for reprogramming of the local terminal from a remote host computer.

FIG. 1A depicts a local terminal sold by Veri-Fone, Inc. of Honolulu, Hawaii, under the trademark ZON. The terminal 10 comprises a base unit 11 and a telephone handset 12. The base unit 11 includes an alphanumeric display 13 having sixteen characters, a keyboard 14, a cardreader 15, a speaker 16, and a speaker volume control 17. The handset 12 includes a handset ringer switch 18, transmitter mouthpiece 19, a hang-up button 20, and a telephone keypad 21. The telephone keypad 21 is optional since, as will be discussed below, the telephone dialing function can be accomplished using a specific section of keys on the keyboard 14. The handset 12 is connected to the base unit 11 by way of a standard cord 22 and a standard jack 23.

The back (not shown) of the base unit 11 has a modular telephone jack for connecting the unit to a telephone outlet. It also has a plug into which an optional P.I.N. (Personal Identification Number) Pad may be connected, a serial port connector for a printer, and a plug to connect to a power pack for supplying electrical power to the unit. The base unit houses the electronic circuitry of the terminal which is shown in block diagram form in FIGS. 2 and 3.

Referring to FIG. 1B, it is seen that keyboard 14 has three sections of keys designated 14A, 14B, and 14C, comprising a total of twenty-eight keys in all. Each of the sections of keys serve multiple functions which depend on the state that the terminal is in. The keys in all three sections serve as alphabetic letter input keys with the particular letter assigned to each key shown as a label above the key. The keys in section 14A serve also as numeric input keys and transaction host selection keys. Note the identifiers of the major credit cards next to keys having the numeric labels 1, 4, 5, 7, and 8. The keys in section 14B serve as calculator function keys and the keys in section 14C serve as terminal function keys.

The standard functions that the ZON terminal performs are standard and programmable telephone dialing operations, four function calculator operations and point of sale transaction terminal operations.

a. The Telephone Function

Standard telephone dialing may be performed by using the keypad on the telephone handset. In this mode, the handset is directly connected to the telephone line when the handset is lifted and the keyed in tones are generated by the tone generation circuitry within the handset as the individual keys on the handset are pressed.

Telephone number dialing may also be done as a programmed feature of the terminal with the handset in place. When the "PHONE" key in section 14C of the keyboard 14 is depressed, the terminal enters the "phone" state and the keyboard section 14A becomes a telephone keypad for a computer telephone function. In this state, each of the keys labelled 0–9 invokes a DIAL-DIGIT action which is a predefined read only memory resident task of the computer system within the terminal and causes an appropriate tone to be generated and put out over the telephone line. It also causes each of the dialed digits to be stored in sequence in a reserved memory location for accessing using the REDIAL key in section 14A. This function of the terminal is an integral part of the application program which is executed by the terminal as will be discussed below in connection with the contents of a generic program which has been prepared for the ZON terminal using the method of this invention. These particular individual functions are predefined ROM functions in the ZON terminal and are accessed from the state tables, but it should be understood that they could also be application program module functions defined in the application program itself.

The terminal also has a memory speed dialing feature which permits up to eight frequently used telephone numbers to be programmed into the terminal, stored in memory in the terminal and later automatically dialed by the computer system within the terminal. The details of the programming of these phone numbers is not important to this invention and will not be discussed here. Once the numbers have been stored, the memory speed dialing is accessed by sequentially depressing the PHONE and RECALL keys in section 14C of the keyboard and then entering a two digit number between 09 and 16 on the keypad section 14A. The computer within the terminal produces in sequence the tones corresponding to the digits in the stored phone number. This is a standard speed dialing function which is available in many type of computer-based telephones and need not be described in detail here. These functions can be altered by altering the application program, for example, by altering the number of phone numbers that can be stored, or by altering the particular keys on the keyboard that are utilized to access the features. The terminal is programmed automatically to return to the READY state when the handset is replaced. This can also be changed or redefined by the application program.

b. The Calculator Function

The calculator function is accessed from the READY state of the terminal by pressing the CALC key in section 14B of the keyboard 14. The display responds by displaying "CALC", and the keyboard sections 14A and 14b then become number entry and math function keys of a regular four-function calculator. The functions of the calculator feature of the terminal are also predefined ROM resident functions accessed from the state tables, but these functions could alternatively be defined in the application program itself.

c. The Transaction Terminal Functions

The basic transaction terminal function is credit verification related to a sale using the customers credit card and either the manual or credit card reading method of entering the credit card data. Table V attached hereto illustrates the operation of the terminal in a typical transaction using the credit card method. The keystroke which selects the host automatically sets up the terminal to dial an appropriate telephone number to access the host computer for the credit card that is being used. In response to the display prompt to enter the account number, the credit card is swiped through the reader. If the information on the magnetic stripe of the card is successfully read by the terminal, the display prompts for the entry from the keyboard of the amount of the sale, followed by depression of the ENTER key. Once the ENTER key is depressed, the terminal takes over the processing of the transaction. The host computer is dialed, the transaction information is transmitted over the phone line, the approval information is received from the host computer, the approval information is displayed to the operator for noting on the credit slip and the terminal disconnects. Depressing the HANG UP key terminates the function and returns the terminal to the READY state.

The manual entry method shown in Table VI is utilized if the magnetic stripe on the customer's credit card is damaged. The transaction proceeds in the same manner except for manual entry of the credit card number and the expiration date of the card in response to prompts on the display of the terminal.

The terminal is also programmed to enable recall of the information transmitted and received during the last transaction carried out on the terminal after the HANG UP key is depressed. Another feature which is provided in the terminal is access to use of the McDonnell Douglas electronic draft capture system for capturing information on transactions in a batch processing mode. Data on sales, voids, and credits can be entered as well as doing post transaction authorizations, transaction batch inquiries, batch reviews in the forward or reverse direction, and other standard functions of the McDonnell Douglas system.

d. Standard and Custom Terminal Features

It should be understood that, using the method of this invention, the features which are provided as standard features of the terminal can be determined by the terminal vendor as part of its standard product strategy. Various standard models having different features can be offered by using somewhat different application programs for the different features of the different models. The terminal can also be customized in its function by providing a custom application program which adds other features to the standard ones provided or alters the features of the terminal or a combination of the two. Each of the following general functional areas of the terminal can be customized:

** data entry
** packet communication
** data capture
** local processing of data
** unassisted batch data transfer
** report generation and printing
** receipt slip formatting and printing
** PIN entry for identity verification

The terminal can be custom programmed for control of external devices, such as the PIN pad for entry of personal identification number and printers which may be attached to the terminal to print receipts or reports, or control of other devices which can communicate with the terminal over the RS232C interface.

2. Illustrative Example of Local Terminal Hardware Environment for the Invention (FIGS. 2 and 3)

A particular local terminal hardware environment in which the method of this invention may be practiced is illustrated in the block diagrams of FIGS. 2 and 3 and will now be described. The hardware illustrated in that of the ZON terminal depicted in FIG. 1. However, it should be understood that the method of this invention is a general one and can be applied in a number of different local terminal hardware environments having at least a minimum set of hardware elements required for use of the invention.

FIG. 2 taken together with FIG. 1 illustrates generally one example of a hardware system environment in which the method of this invention may be practiced. Local terminal 10 includes a microcomputer system 30 which incorporates memory and input/output circuits which communicate with and control speaker 16, card reader 15, display 13, and keyboard 14. Optional PIN pad 38 and printer 39 are also controlled and communicated with by the microcomputer system. A communication channel, comprising tone generator 31, modem 32 and telephone interface 33 is provided for communication with a remote computer system 37 via telephone company lines 34. The remote computer system will typically also have a modem 36 and a telephone interface 35 associated therewith. As will be explained in more detail later, this communication channel enables the local terminal to receive a download application program for storage in the memory circuits of the microcomputer system. This same communication channel thereafter serves to enable the local terminal to communicate with remote host computers to perform credit verification functions and/or data capture functions as generally described above.

It should be understood that the remote computer system which performs the application program download to the local terminal is not necessarily involved later in the credit verification function. The local terminal may be communicating with several different host computers for verification of credit relative to different credit cards.

FIG. 3 illustrates the hardware of the ZON terminal in more detail. The overall hardware system is generally quite conventional and the details of its components and the functioning of each will be apparent to persons knowledgeable about local terminal hardware. Thus these details will not be set forth herein, and only those aspects of the hardware which are pertinent to the

method of operating the local terminal in accordance with this invention will be described.

Central processor unit (C.P.U.) 40 communicates over address, data and control busses 41 with read only memory 42 and random access memory 43. Read only memory 42 stores the operating system program which provides basic functional control of the microcomputer system within the local terminal. In the ZON terminal, a standard Z-80 central processor unit is employed and the operating system is a version of a real-time, multitasking executive called the AMX system available from KADAK Products Ltd. of Vancouver, British Columbia, Canada. This invention is not limited in any sense to this hardware or operating system environment and, as will be apparent to persons of skill in the computer art, the method of this invention may be used in local terminals which utilize a variety of computer hardware systems and realtime operating systems therefor.

Read only memory 42 also stores the terminal software which performs certain of the steps of the method of this invention which are carried out in the local terminal, including the scheduler routine, the command execution routine and the application program download routine. ROM 42 stores the general purpose operation routines which are accessed by the application program modules of the application program. Read only memory 42 may store certain predefined ROM-resident program modules which are standard program modules for performing tasks which are typically included as standard functions of the local terminal. It may also store a default application program having limited features so that the terminal operating system can copy this default program to random access memory 43 in the event that there is a major problem with the integrity of the application program stored in random access memory 43. In addition ROM 42 may store the following program components:

** data entry routines
** a control string interpreter
** a file manager system
** a direct download program
** miscellaneous support routines.

or any other functions which are to be made available under the control of the application program.

Random access memory 43 stores the application program modules which are downloaded from the remote computer along with other information forming the operational environment of the application program. Random access memory 43 also has a section which serves as scratch pad memory, including memory sections which may serve as data registers and buffers. Details of the types of information stored in random access memory 43 will become apparent from a consideration of the detailed description below of the information which is downloaded as part of the application program.

Random access memory 43 is preferably provided with battery backup to preserve the contents of the memory which are otherwise volatile in the event of loss of power to the local terminal. Another option is to use non-volatile random access memory circuits and associated system components which provide the capability to detect an impending power failure and to copy the contents of the volatile portion of the random access memory circuits to a non-volatile component of the circuits before all power to the local terminal is lost.

GENERAL METHOD OF THIS INVENTION

1. The Core Programming Methodology

The method of this invention includes three initial steps which form the core methodology of the invention and which are the basis for achieving the advantages of the invention which are summarized above. These three steps comprise the following:

(a) Establishing a set of general purpose operation routines to be executed by the local computer system in the local terminal, with each of these routines comprising a set of instructions for execution by the central processor unit in the local terminal in a prearranged manner to accomplish a specific task;

(b) Defining a set of commands, each of which is associated with a specific one of the general purpose operating routines and includes at least an operation code relating the command to the associated general purpose operation routine, with each of the commands having an associated command code length substantially less than the code length of the associated general purpose operation routine; and

(c) Storing the set of general purpose operation routines in the read only memory of the local terminal.

a. Establishing the General Purpose Operation Routines

The step of establishing a set of general purpose operation routines involves identifying all of the specific tasks which the local computer system in the local terminal may be called upon to perform in the particular type of application to which it will be devoted. Once these tasks have been identified and fully defined, it is a relatively straightforward, though not necessarily trivial, task to create the set of routines which will accomplish these tasks. The routines must include the capability of performing all tasks which are necessary for the types of application programs to be executed by the terminal.

Preferably, the general purpose operation routines will be created in machine language which can be assembled into machine instructions which, in turn, can be directly executed by the central processor unit in the local terminal. This central processor unit will typically be a microprocessor having a specific machine instruction set. Utilizing machine language programming and having such programs authored by a highly skilled machine language programmer will provide the fastest execution time for the individual general purpose operation routines and thus faster execution time for the application program running at the local terminal.

It might also be possible to use a higher level language, such as the FORTH computer language, instead of machine language instructions. In such a case the execution of the instructions of the higher level language will require the intervention of an interpreter. If this does not result in too great a sacrifice of execution speed, the higher level language approach may be satisfactory. However, since the method of this invention inherently involves trading off speed of execution of the application program for reduced application program download time, use of expertly authored machine code for the general purpose operation routines is highly preferred.

Table I appended hereto sets forth a listing of the functions to be performed by a set of general purpose operation routines which have been created for the ZON family of terminals described above. The descrip-

tion given in Table I of the functional task to be performed by each of the general purpose operation routines is sufficient for a skilled machine language programmer to author a machine code listing which will accomplish that task. Flow charts of illustrative examples of the general purpose operation routines are included in the drawings and will be discussed below in connection with the step of executing those routines in the local terminal. The illustrative examples together with the descriptions of the specific task to be accomplished by each general purpose operation routine associated with each command illustrates this step of establishing a set of general operation routines sufficiently that skilled programmers can readily adapt the method of this invention for establishing a set of different routines, if required, to be used in other types of local terminal applications.

b. Defining a Set of Commands

(1) General Aspects

The step of defining a set of commands, each of which is associated with a specific one of the general purpose operation routines, will typically be carried out by establishing a high level source code version of each command together with a corresponding object code version of each command and providing program tools for converting the high level source version of each command to the corresponding object code version. This approach to defining commands at source and object code levels together with creation of compiling, linking and assembling tools for conversion from source to object versions is well known in the art. Specific aspects of these tools, which are provided for conversion of the source level commands given in Table I into their corresponding object code, are discussed in more detail below.

(2) Preferred Command Structure and Syntax

The preferred syntax for a command at the source code level is as follows:

{<label>:} <command> {<P1> <P2> . . .
<Pn>} {;<comment>}

where P1, P2 . . . Pn are parameters associated with the command.

The label portion is typically utilized only if the particular application program module in which the command is included requires a label reference for going to that comand line from another location in the application program module. The command name is an integral portion of the command, as is a specification for each parameter which is associated with a command. Comment statements set off by the semicolon are optional. Good programmers provide well commented source code for purposes of documenting the overall functionality of the application program module in which a sequence of commands is provided.

Referring to Table I it is seen that some commands do not have any parameters associated with the command, while other commands have one or more associated parameters. It is also seen that the name assigned to each command preferably is a form of shorthand reference to the task which the associated general purpose operation routine performs. This shorthand reference approach to assigning command names is not a requirement of the invention, but is good practice since it greatly assists the programmers utilizing the set of commands to learn the command set and remember the task associated with each command. As will be discussed in more detail below, each of the command names defined at the source code level preferably converts to an operation code in binary number form at the object code level.

It is the binary operation code at the object code level of the command which relates the command to the associated general purpose operation routine. The manner in which this relationship is established in practice in the ZON terminal application of this invention will be discussed in detail below.

The advantage of a greatly reduced download time for an application program which is achieved utilizing the method of this invention is based on defining each command such that, at the object code level, the associated command code length is substantially less than the code length of the associated general purpose operation routine. Table II appended hereto is a chart which illustrates the substantial reduction in download code size which is achieved using the method of this invention. Table II lists in column one several examples of ZAPD commands found in Table I. In column two the size of the downloaded object code for each command is given. In column 3, the size of the object code level of the machine code routine associated with the command is given. Column four gives an estimate of the minimal machine code size that might be achieved for use in case the entire routine were downloaded in binary object code. Table II illustrates that, in most cases, the ZAPD object code command has a code size substantially less than the code size of the corresponding operation routine.

Table III appended hereto is a chart which shows the set of thirteen parameter types which are utilized in the defined commands shown in Table I. Table IV is a chart which shows a set of thirteen data types which comprise all of the different ways that parameter types may be specified. As shown in Table III each of the parameter types has at least a designated default data type and some of the parameters have alternative data types which may be used to specify that parameter. As shown in Table III, each of the parameter types of byte constant, integer constant, string constant, and address constant have only a default data type. Each of the other parameter types have, in addition to a default data type, two or three alternate data types which may be utilized to specify the parameter.

The default data type is selected for each of these parameter types which have alternate data types on the basis of the data type which is most likely to be used to satisfy that parameter. As will be discussed below, this tends to shorten the overall download code for the application program.

Table IV, in addition to giving the data type reference used in Table III, gives the data type name, the type of identifying symbol which is used at the source code level for specifying the data type reference and an example of a parameter specification utilizing the identifying symbol. In addition, Table IV specifies the length of the parameter in terms of the number of eight bit bytes which a parameter utilizing that data type will occupy in the object code form of the command and the number of bytes that will constitute the value of the parameter during execution of the general purpose operation routine associated with the command. The length specified as "n" means that the length can be up to 255 bytes.

As would be expected, each of the "byte" types of data type references will take on a value of a single byte. Each of the "integer" data types will have a value specified by two bytes and each of the strings will have a value specified by "n" bytes, i.e., up to 255 bytes. The extended address data type does not take on a value since it is typically used to specify a location to which the program may branch under certain conditions.

It should be understood that the parameter type system used in the preferred implementation of the method of this invention as shown in TABLES III and IV is not the only approach that could be implemented. One alternative is to define each command such that the parameters associated with the command can be satisfied by only one data type. This would permit the command parsing module for each command to directly call the parsing routines for the parameters associated with the command without using an intermediate parsing control routine. It would also eliminate the need for the CURVPARM data structure element, the VPARM array, and the data type data elements in the bitmap/opcode system for the object code command syntax discussed below.

However, implementing this alternative approach would require that a much larger number of commands be defined just to handle the different ways that the parameters associated with a particular operation routine are specified in the command. For example, consider the SEND_CHAR command which, as shown in TABLE I, utilizes two X type parameters in the preferred approach to parameter specification. The X type parameter is a varying parameter type with four different data type satisfiers. To cover all of the different ways to satisfy two X type parameters with different data types with different commands could potentially require sixteen different commands, within a family of commands that would all use the same operation routine but would require different parameter parsing routines.

Some of the commands listed in TABLE I use four or five varying parameter types. Using the alternative approach of defining commands with only one fixed set of data types to satisfy the associated parameters could drastically proliferate the number of commands which would be needed to provide the programmer with a reasonable level of flexibility in satisfying the parameters required for a given operation routine to perform its task. To make this approach practicable, the command definition step would necessarily leave out some possibilities, because programmers would not be able to deal with that many different commands and keep them sorted out. Moreover, the maximum number of commands which could be specified by a single byte operation code would be rapidly exceeded, making sixteen-bit operation codes necessary.

It should be noted that this alternative approach to implementation would not alter the number of operation routines that would need to be established, but it would multiply the number of general purpose operation routines to be established in the sense that each of the different commands in each family of commands would require a different command parsing module. Proliferation of the number of commands would greatly increase the number of groups of opcodes that would be required, and thus increase the size of the opset pointer table and opcode pointer table to be stored in ROM. It would also dramatically increase the number of command parsing modules to be stored in ROM, although it would eliminate the parsing control routines. Overall it would make the programming language much more cumbersome to handle.

It is thus seen that the preferred system of using parameter types which include varying parameter types which can be satisfied with plural data types provides a powerful and elegant way to achieve full programmer flexibility in satisfying parameters required by a particular operation routine without proliferation of commands. The programmer needs only master the source symbol notation for the different data types to use in the source syntax of the command and the data types associated with the different parameter types. In connection with this, the information shown in TABLES III and IV includes a new alternate form of notation which is different from the form used in TABLE I and which will eventually make the whole structure of this parameter system more logical and thus easier to remember.

(3) Object Code Syntax of Defined Commands (FIGS. 4A and 4B)

Referring back to Table I, it can be seen that, in this particular implementation of the method of this invention, each of the commands in the defined set of commands is assigned an opcode group and an opcode number. The opcode group and opcode number set forth in Table I for each command is given in a decimal notation. In the object code command syntax, the opcode group is represented in binary form as 2 binary digits ("bits") in a bitmap byte. The opcode number is represented in the object code command syntax in the form of 7 bits of an opcode byte. Thus, in this version, the operation code which is defined for each command in the set of commands is a combination of a binary opcode group value, also designated "opset", and an opcode number, designated "opcode".

The purpose for this approach can briefly explained as follows. The Z80 micro processor utilized in this terminal is an 8 bit micro processor. Accordingly, the object code version of an application program module is limited to use of 8 bit bytes. An 8 bit byte can take on 256 different values, but with one bit reserved in this implemetation for a byte type flag, only 7 bits are available for designating the opcode. Thus, in order to provide the capability of having more than 128 commands and more than 128 associated operation codes, the opcode group and opcode number scheme was adopted. A two-bit opset number provides for four opcode groups and a seven-bit opcode number provides for 128 different opcodes in each opcode group. Together they provide the capability of having up to 512 total commands and associated operation codes.

FIGS. 4A and 4B illustrate the operation code system which has been adopted for implementation of this invention in the ZON terminals. This embodiment of the operation code system utilizes a combination of opcode bytes and bitmap bytes as shown in FIG. 4A Each opcode byte and bitmap byte has a bit 7 representing a byte type flag and the value of that type flag distinguishes opcode bytes from bitmap bytes. If the type flag has a binary 0 value, the byte is an opcode byte. If the type flag has a binary one value, the byte is a bitmap byte. The remaining seven bits of each opcode byte represents the binary opcode which can take on one of 128 different values.

The low-order six bits of a bitmap byte, i.e. bits 0–5, represent three groups of "data type" data elements. Each of these data elements is used for specifying the

data type used by the programmer in the source code level of the command to satisfy the parameter. Since each parameter type which takes on varying data type values has at most four possible data types associated with it, two bit data elements are sufficient to specify the data type. Table III shows the binary values which are used in the data type data elements of the bitmap byte for designating the data type of the parameter.

The sixth bit of each bitmap byte is an opset bit and is used to designate the value of one of the two bits utilized to specify the opset in which the command associated with the particular operation code is grouped. Since there is only one opset bit in each bitmap byte, the operation code for each command may need to utilize two bitmap bytes to designate the opset into which the particular command is grouped. It will be seen, however, that for commands which are grouped in opset 0, one or both of the bitmap bytes may be "implied" bytes that are not actually present in the object code command syntax. For commands in opset 1, one of the bitmap bytes may be "implied". In other words, 1 or 2 bitmap bytes may be eliminated, depending on the number of non-default type varying parameters which are specified by the programmer in using the associated command. FIG. 4B illustrates this operation code bitmap and opcode system more extensively.

As shown in FIG. 4B, the object code command syntax for a command which falls into opset 00 may have one of three different structures. All commands in opset 00 which have no varying type parameters can be represented in object code syntax by a single opcode byte followed by whatever parameter bytes are required. In addition, any command in opset 00 which has parameter types which may have varying data types, and for which all of the assigned data types are the default type, may be represented by a single opcode byte followed by bytes comprising the parameters themselves. In both of these cases, no bitmaps are required since the opset 00 is the default opset. As will be shown in other Figures discussed below, the "translator" program uses an opset data structure which has a default value 00. Another way of looking at it is that the operation code for the command contains two implicit or understood bitmap bytes wherein the opset bit in each of the bitmap bytes, and all of the bitmap "data type" elements have value 0. Since there is no need for actual bitmap bytes, there is no reason for the compiler which generates the object code version of a command with no varying or non-default parameters to include any bitmaps.

The second form which the object code command syntax may take for commands which are grouped in opset 00 combines a single bitmap byte with a single opcode byte followed by whatever parameter bytes are required. A single bitmap byte permits up to three non-default varying parameters to have their data types specified in the three data type data elements which are provided in a single bitmap byte. The opset bit in the single bitmap byte must have a 0 value in this case to correspond to the first bit of the two-bit opset specifier. In this case the operation code for the command has one bit of the two-bit offset specifier understood, i.e., one of the bitmaps is implied or understood and not physically present.

The third form of object code command syntax for commands which fall into opset 00 is shown in FIG. 4B as involving two or more bitmap bytes followed by an opcode byte and whatever parameter bytes are required. With two bitmap bytes actually present, up to six non-default data types can be specified for parameter types which can take on varying data types. It is seen that the opset bit in each of the bitmaps has a 0 value and these two digits together indicate opset 00.

As shown in FIG. 4B, commands which are assigned to opset 01 may have an object code command syntax in one of two different forms. The first form involves a single bitmap byte and the second form involves two bitmap bytes. In the single bitmap byte case, the opset bit must have a value 1 to designate that the associated command has been assigned to opset 01. If two bitmap bytes are present, the opset bit in the first must have a 0 value and the second must have a 1 value so that together they designate opset 01. As seen the first form of command syntax with a single bitmap byte can handle up to three non-default varying parameters and the second form can handle up to six non-default varying parameters.

Two additional opcode groups, namely groups 2 and 3 having opset values 10 and 11, respectively, are provided for in the operation code system utilized in the embodiment of this invention implemented in the ZON terminal, but none of the defined commands have been assiged to either of these opcode groups. Opset 10 and opset 11 each have an object code command syntax which requires two bitmap bytes to specify the two bits which designate the opset. In other words, two bitmap bytes must be present regardless of whether the command has non-default parameters specified.

It should be understood that the operation code bitmap and opcode system illustrated in FIG. 4B could use more bitmap bytes in the command syntax than is shown. In the version of the invention implemented in the ZON terminal, provision is made for up to four bitmap bytes. This provides for up to twelve varying type parameters to be included in a command and have data types specified for each.

To minimize the code size of the object code form of the defined commands, all commands which have associated parameters of a type which have only a default data type may be grouped in opset 00 since such commands will, in every case, require no bitmap byte. Similarly, all commands which have no associated variables may be assigned to opset 00 since no bitmap bytes would be necessary for the object code version of such commands. This approach is obviously not necessary to implementation of the invention.

Another technique which may be used to reduce the size of the object code form of the commands which are included in an application program involves reordering the position of the parameters, in the ordered list of parameters defined as part of the command, as between the order designated in the source level, i.e., the source parameter syntax, and the order designated in the object code, i.e., the object syntax. Referring to the command set in Table I, it will be noted that, in some cases, the source syntax and the object snytax of some commands having more than one parameter may be different.

The purpose for reordering the parameters as between source and object syntax is to place those parameters which have only a default data type, and/or which are most likely to be assigned a default data type, at the end of the ordered list of parameters in the object syntax. This will tend to reduce the number of bitmaps present in the object code of the application program since data type data elements in bitmap bytes are only required to designate non-default data types for varying

parameters. This is not a requirement of the operation code bitmap and opcode byte system used to implement the method of this invention, but it is relatively easy to provide this feature in the compiler which converts the source code level of the commands to the object code form thereof, and it provides further compression of the size of the object code to be downloaded.

(4) Alternative Operation Code Systems and Associated Object Code Command Syntax (FIGS. 5-8)

It should be understood that the method of this invention is not limited to the operation code system described above and other approaches to defining commands which include an operation code relating the command to an associated general purpose operation routine could be provided. For example, consider a case in which up to 256 commands may be defined and none of the commands requires more than four non-default varying parameters. In such a case, the operation code system could utilize the object code command syntax shown in FIG. 5A.

In this system, each command at the object code level would have a single bitmap byte and a single operation code byte followed by necessary parameter bytes. Since there is no provision for a type flag in either the bitmap or the operation code byte, both bytes must always be present at the object code level, so that the two bytes can be distinguished by their position in the multi-byte command sequence. Obviously, the order of the bitmap and operation code bytes could be reversed from that shown in FIG. 5A.

FIG. 5B shows another system in which two bitmap bytes are present in the object code command syntax along with a single operation code byte. The approach shown in FIG. 5B would permit up to eight varying parameters to be designated with different data types. Again both bitmaps would always have to be present, and the command execution program running in the local terminal would distinguish the bitmap bytes and the operation code byte on the basis of position in the object code command syntax. Other systems with more bitmap bytes to provide for more data type data elements for varying parameters could also be used.

FIG. 6 illustrates another version of an operation code system which may be utilized in versions of this invention in which varying type parameters are not used. In this case, bitmap bytes are not utilized and two operation code bytes are provided. The operation code bytes may, for example, take on values corresponding to ASCII characters with two characters designating an operation code. This system is actually used in one alternative implementation of this invention described below. It should be apparent that the two opcode bytes could also designate corresponding commands and related operation systems in accordance with other notational approaches.

FIG. 7 illustrates an operation code system which might be employed in a local terminal system which utilized a central processor unit operating on the basis of sixteen bit words, rather than bytes. In such a system, eight bits of the word could be designated as the operation code and the other eight bits designated as 2-bit data elements for varying parameters. In this case the position of the bits within a single word designates what the bits represent.

FIG. 8 shows another alternative operation code system which might be employed in implementing the method of this invention. In the FIG. 8 system, the operation code is a sixteen bit code. This sixteen bit code may be utilized to directly address the associated general purpose operation routine or a pointer to the general purpose operation routine.

(5) Compiling Source Code Commands into Object Code Command Syntax

Based on the above description of the operation code bitmap and opcode system, persons of skill in the programming art readily appreciate that the conversion of the defined commands at the source code level to the object code command syntax involves a relatively simple compiling function. Basically, the compiler converts the compound name and the assigned parameters into a set of 1 to 5 numeric values representing in decimal number format the binary values of bitmaps and opcodes. These decimal numbers are followed by the actual parameters associated with the command. As shown in the example of the generic transaction terminal program in Tables XXX and XXXI the compiler converts the source code command syntax into a sequence of code statements which can then be operated on by a machine language assembler and linker program to convert the code statements into binary form for the download into the local terminal. The final binary object code form of the generic program is shown in Table XXXII. The process of conversion from source to object code will be discussed in more detail below.

c. Storing the General Purpose Operation Routines

(1) The Preferred Storage and Accessing System (FIG. 9)

FIG. 9 illustrates one approach to storing and accessing general purpose operation routines based on the opset/opcode system for designating the operation code of the associated command. As shown in Table I, most of the commands include a command name which is converted to an operation code and also a set of parameters with one or more parameter members in the set. Since this implementation of the method of this invention provides for parameters of varying data type, the general purpose operation routine associated with each command which includes parameters which may take on varying data types must have the capability of first "parsing" the parameters associated with the command and then executing an operation routine utilizing the parsed parameters.

For purposes of conceptual clarity, the general purpose operation routine associated with each command will be considered to incorporate the parameter parsing routines required for parsing the parameters associated with the command. This incorporation may be accomplished directly, i.e. by actually including the necessary parsing routines with the operation routines. Alternatively, and preferably to save memory space in the local terminal, the necessary parameter parsing routines are incorporated by reference to a set of parsing routines which are global to and shared by the different operation routines. The storage and access system shown in FIG. 9 illustrates an approach to storing parsing routines separate from operation routines and incorporating parsing routines by reference into general purpose operation routines.

As shown in FIG. 9, one implementation of the method of this invention involves storing in the read only memory of the terminal an opset pointer table, an opcode pointer table, a set of command parsing mod-

ules, a set of parsing control routines, a set of data type parsing routines, and a set of operation routines. The opcode pointer table is divided into separate opset tables and each opset within the opcode pointer table has a pair of pointers associated with each opcode value. The first pointer is a parsing module pointer and the second is an operation routine pointer. The parsing module pointer contains the address of one of the command parsing modules, i.e. the particular command parsing module associated with the command to which this opcode has been assigned. The operation routine pointer contains the address of the associated operation routine. Each of the command parsing modules includes program code for calling in sequence one or more of the parsing control routines. Each of the parsing control routines includes program code for calling one data type parsing routine from a subset of the total group of data type parsing routines. This will be explained in more detail below.

The opset pointer table contains three bytes of information, the first two bytes being an opset pointer and the third byte designating the maximum number of opcodes that have actually been defined in that particular opset. As will be discussed below, the MAX.OPCODES byte in the opset pointer table is used to verify whether the opcode number in the command being executed is a valid one, that is whether it falls within the range of opcodes actually utilized in the system. The opset pointer for each of the opset groups contains a two-byte address which is the address of the start of the corresponding opset section of the opcode pointer table.

As shown in FIG. 9, there is an algorithm which is executed to locate the appropriate opset pointer in the opset pointer table and another algorithm to calculate the location of the parsing module pointer. From the binary object code version of the command, the command execution routine recovers an opset value and an opcode value and stores these in data structures as will be described below. The location of the opset pointer may be calculated by adding three times the binary value of OPSET to the starting address of the opset table. Since there are three bytes associated with each opset in the opset pointer table, this algorithm, in effect, calculates the address of the opset pointer. When the opset pointer has been retrieved from this address, the address of the parsing module pointer is calculated by adding to the two-byte address of the opset pointer a value of four times the opcode value recovered from the binary version of the command.

Once the address of the parsing module pointer has been calculated, the two byte address which is stored at the calculated parsing module pointer address can be loaded and used to call the associated command parsing module. The subsequent call to the operation routine can be accomplished by utilizing the two-byte address which comprises the operation routine pointer immediately following the parsing module pointer. It is thus seen that the opcode pointer table directly associates a command parsing module with an operation routine. Together the command parsing module and the routines which it calls and the operation routine itself comprise the general purpose operation routine for each command.

(2) Alternative Storage and Accessing Systems (FIGS. 10 and 11)

The purpose for using the preferred scheme discussed above for incorporating a parsing routine and operation routine by reference into an overall general purpose operation routine associated with each command is to reduce the amount of memory required for storing general purpose operation routines in the read only memory of the local terminal. It should be apparent that a trivial alternative, albeit one which would consume substantial additional memory space, would be to include all of the parsing control routines and data type parsing routines required for a particular command in the actual general purpose operation routine associated with that command. FIG. 10 illustrates such a system.

In this case the opcode pointer table need only contain a two-byte address of the corresponding general purpose operation routine. It should be apparent that this is a much less desirable approach from a memory space standpoint since some commands have as many as five or six associated parameters and several of those parameters may be varying data type parameters. Including all of the parameter parsing routines required in such a case locally in the general purpose operation routine would dramatically increase the code size of the general purpose operation routine and thus substantially affect the cost of memory in the terminal. It should be noted that this would have no affect on the application program download time and would have little or no beneficial affect on the speed of execution of the individual commands of the application program.

Going back to FIG. 9, the reason for separately storing command parsing modules relates to the circumstance that a number of the commands which are defined and set forth in Table I share the same ordered list of parameter types. For example, there are several commands having a single parameter of the same parameter type. There are other commands which have two parameters of the same types in the same order, and therefore have the same object syntax. Thus memory space is conserved by having the general purpose operation routines share command parsing modules.

However, as another alternative approach which would not consume a large amount of additional memory, each of the general purpose operation routines might include its own command parsing module of the type required for calling the parsing control routines which, in turn, call the appropriate data type parsing routines. This alternative scheme is shown in FIG. 11. In this case the opcode pointer table would contain the address of the general purpose operation routines and a separate parsing module pointer would not be required. The command parsing module would be located at the starting address of the general purpose operation routine which is pointed to by the operation routine pointer in the opcode pointer table.

Referring back to FIG. 9 in conjunction with Tables III and IV, it should be understood that the number of data type parsing routines stored in read only memory in the local terminal corresponds with the number of specific data types set forth in Table IV. In other words, in the present implementation of this invention in the ZON terminal, there are thirteen data types and each of these data types has a separate data type parsing routine associated therewith. As shown in Table III, there are twelve parameter types and each of these parameter types has a parsing control routine associated therewith.

Those parameter types which have only a default data type have a parsing control routines established such that it will call only a single data type parsing routine, namely the one associated with the single data type assigned to that parameter type. Others of the

computer system via said communication channel, including the step of storing for each of said set of commands said operation code associated with said prearranged sequence of commands in each of said application program modules of said preselected application program in said random access memory; and

f. establishing in said local computer system a program execution routine for enabling said central processor unit to execute a selected one of said application program modules in response to said input means by repetitively performing the steps of:

(1) reading said associated operation code associated with said selected application program module stored in said random access memory;

(2) accessing a specific general purpose operation routine associated with said associated operation code; and

(3) executing said specific general purpose operation routine.

4. The method of claim 1, wherein said step a. includes the steps of:

(1) establishing as a first one of said general purpose operation routines a START PROGRAM routine comprising machine instructions for calling for execution of a specified application program module;

(2) and establishing as a second one of said general purpose operation routines a WAIT routine comprising machine instructions for suspending execution of an application program module in which an operation code corresponding to said WAIT routine appears;

step d. comprises establishing a plurality of application program modules each comprising a sequence of said operation code, at least a first one of said application program modules including an operation code associated with said START PROGRAM routine for calling for execution of a specified second one of said application program modules, and at least one of said first and second application program modules including an operation code associated with said WAIT routine;

said method further comprising the steps of:

g. defining a program data structure format comprising a plurality of data elements associated with execution of an application program module by said program execution routine;

h. establishing in said random access memory an array storage area for storing a program data structure array comprising a plurality of application program module slots each having said program data structure format;

i. establishing in said random access memory an area for storing scheduler data program structures comprising a WAIT data element for storing define "wait" or "continue" data values, and a plurality of ACTIVE PROGRAM data elements each associated directly with one of said program slots in said program data structure array and storing one of two defined data values indicating whether or not said associated program slot contains data elements associated with an active application program module;

j. establishing in said local computer system a program scheduling routine for enabling said central processor unit to carry out the step of scheduling the sequential execution of application program modules one at a time based on the data values stored in said ACTIVE PROGRAM data elements, said execution of each active application program module continuing until said WAIT data element contains a wait data value or until the currently executing application program module has completed its execution;

said START PROGRAM routine including instructions for loading one of said ACTIVE PROGRAM data elements with a data value indicating presence of an active application program module and for loading into said array storage are of said random access memory a corresponding program data structure associated with said application program module called for execution by said routine; and

said WAIT operation routine including instructions for storing said "wait" data value in said wait data element of said scheduler data structure.

5. The method of claim 4, wherein said program data structure array established in step h. is a fixed array with a fixed number of application program module slots;

said ACTIVE PROGRAM data elements in said scheduler data structure established in step i. comprise an ACTIVE PROGRAM fixed bitmap data structure with each bit location in said bitmap corresponding to as associated application program module slot in said program data structure array and storing one of a first bit value or a second bit value indicating the presence or absence, respectively, of an active application program module in the associated application program module slot;

said START PROGRAM routine includes a parameter specifying which application program module slot should be utilized for storing data structures for execution of said associated application program module.

6. The method of claim 5, wherein said wait data element established in step i. may also store a defined "HALT" data value; said step a. further includes the step of:

(3) establishing as another one of said general purpose operation routines a DONE routine comprising instructions for loading said wait data element with a HALT data value and loading the bit location in the ACTIVE PROGRAM bitmap of the application program module which includes said DONE routine with said second bit value indicating the absence of an active application program module in the associated application program module slot;

and each of said application program modules which completes execution without branching to another application program module includes an operation code corresponding to said DONE routine as a last command therein.

7. The method of claim 6, wherein

said scheduler data structure established in step i. further comprises a BACKGROUND bitmap having bit locations corresponding to application program module slots in said program data structure array established in step h. and storing one of first or second bit values indicating whether the active application program module in a corresponding bit location in the ACTIVE PROGRAM bitmap has a background or foreground attribute;

parameter types have three or four possible data types associated therewith. Accordingly the parsing control routines associated with those parameter types are established to determine the actual data type and call one of three or four of the data type parsing routines. Thus each of the twelve parsing control routines has associated therewith a subset of the data type parsing routines with the subset comprising one, three or four data type parsing routines.

The number of command parsing modules corresponds to the total number of different ordered lists of parameters in the object code syntax for the totality of the commands which are defined relative to the general purpose operation routines which are established. Although this means that there are large number of command parsing modules, the amount of code required to execute a sequence of calls to different parsing control routines is relatively small. Consequently, the amount of memory space occupied by the large number of command parsing modules is not overly extensive.

Referring back to FIGS. 5-7 which depict alternative operation code systems, if a single 8 bit operation code as shown in these alternative examples is utilized, a single opcode pointer table is all that is required. The parsing module pointer location can be calculated by adding four times the values of the operation code to the starting address of the opcode pointer table.

Referring to the operation code system shown in FIG. 8, the sixteen bit operation code could be used to directly address the parsing module pointer in a single opcode pointer table. If the operation code system depicted in FIG. 8 were utilized in conjunction with the approach to storing the general purpose operation routine which is depicted in FIG. 11, the sixteen bit operation code could be used to directly address the associated general purpose operation routine and the opcode pointer table could be eliminated.

These variations in implementation of the three initial steps which form the core methodology of this invention illustrates that the method of this invention is a general one. The invention is not limited to any particular form of implementation of the core programming methodologies, which provide a substantial reduction in the code size of an application program to be downloaded from a remote computer into a local terminal where it will be executed. The degree of application program code compression and the corresponding degree of reduction in application program code download time will, of course, be somewhat dependent on the particular implementation chosen.

2. Establishing an Application Program Module

Having established and stored the general purpose operation routines so that they can be executed at the local terminal and having defined a set of commands related to those general purpose operation routines, the next step is to establish an application program module on a remote computer system. This application program module will comprise a prearranged sequence of the commands which have been defined and will thus direct the local terminal to carry out a sequence of tasks by executing the associated general purpose operation routines in a particular order. Typically an overall application program to run on the local terminal will comprise a number of different application program modules, each of which enables the local terminal to carry out a set of functions associated with the transactions or other operational features which are desired.

The step of establishing an application program module is carried out by a programmer, utilizing the defined command set to put together a meaningful ordered sequence of commands, including designated parameters for each of the commands, so that a meaningful function is defined by the application program module. Other aspects of establishing an overall application program include establishing the environment of the overall application program, such as establishing application program variables and files which will be utilized. All of these are relatively standard aspects of programming. A specific illustrative example of steps involved in establishing an overall application program, utilizing the specific implementation of this invention which has been created for the above mentioned ZON terminals, and utilizing the command set which is illustrated in Table I, will be set forth in detail below.

3. Communicating the Application Program Module to the Local Terminal

The next step of the general method of this invention is to communicate the application program module which has been created from the remote computer to the local terminal. This communication step, referred to general as downloading the application program, involves the use of a download program routine operating at the local terminal, a download program operating at the remote computer and a communication link between the remote computer and the local terminal. In general, this communicating step or downloading step involves communicating a copy of the binary object code version of the overall application program, including all of the individual application program modules established in the prior step of the method, into the random access memory of the local terminal. It may also involve communicating ASCII data, the meaning and usage of which is defined by the application program, into the random access memory of the terminal.

There are numerous ways that this step can be accomplished, and a specific discussion of the application program downloading methodology which has been implemented for use in the ZON terminal is given below.

4. Establishing a Program Execution Routine

The next step in the general method of this invention is to establish in the local computer system a program execution routine for enabling the central processor unit in the local terminal to execute the application program module. This execution of the application program module is accomplished by repetitively performing the steps of reading one of the operation codes stored in the random access memory at the local terminal, accessing a specific one of the general purpose operation routines associated with the operation code which has been read, and then executing the accessed general purpose operation routine. As discussed above, the general purpose operation routine may include parsing routines for parsing the variables which are included along with the operation code in each command. To illustrate this step of the method, a specific example of methodology utilized to execute a general purpose operation routine associated with one command of an application program module will be discussed.